
OM03-0226-1

OMCS/RCX Report Content indeXer

Industrial Grade Report Data Mining for OS/390 and z/OS platforms

USER'S GUIDE

OMCS/RCX for OMCS Version 2.8

Revised - May 2007

LICENSED MATERIAL

COPYRIGHT Australian Systems Engineering Pty. Ltd. 2003,2004,2005,2006,2007

This material always remains the property of Australian Systems Engineering Pty. Ltd. and must be used only in accordance with the terms of the agreement under which it is supplied.

Manual and Product identifiers.

This manual is identified by a publication number, **OM03-0226**, combined with a revision level number (sometimes called a dash level) **-1**.

The licensed software product, of which this manual is a part, is identified by a **Product Name** or **Product Acronym** and an **Edition Number**. Edition numbers are made up of a **Version number** and a **Release number** within that version.

The material in this manual is intended to be used with **OMCS Report Content indexer Version 1 Release 2** and later editions of that product. The edition number may appear in licensed material in abbreviated forms like the following: **V1R2**, and **V1.2**.

From time to time ASE may issue replacements for, or additions to, the pages in this manual.

Each time the entire manual is replaced by a later version then the revision level (the dash level) part of the publication identifier will be increased by 1.

Related Publications

None

Published by

Australian Systems Engineering Pty. Ltd.
136 Byrne Street
Fitzroy North, Victoria 3068, Australia
telephone +61 3 9489 4033

email: help@ase.com.au

Trademarks

OS/390 and z/OS are registered trademarks of IBM Corp.

UNIX is a registered trademark of The Open Group

Contents

Contents	3
INTRODUCTION	7
<i>Exploit your enterprise's existing application reports resource.....</i>	<i>7</i>
<i>RCX Components:</i>	<i>7</i>
<i>Pre-requisite software.....</i>	<i>7</i>
RCX SETUP	9
DATABASE REQUIREMENTS AND SIZING	9
<i>Four databases are required.....</i>	<i>9</i>
<i>Database naming and usage notes</i>	<i>9</i>
ALLOCATING THE DATABASES	10
<i>Using ISPF 3.2</i>	<i>10</i>
<i>Using the sample allocation JCL:</i>	<i>10</i>
SETTING GLOBAL RCX PARAMETERS	10
<i>Invoking the RCX Administration ISPF option.</i>	<i>10</i>
<i>Admin option 2: Initialise Application Control Files.....</i>	<i>11</i>
<i>Admin option 1: Maintain OMCS/RCX Application Controls.....</i>	<i>11</i>
<i>Admin option 1.1: Global Settings</i>	<i>12</i>
<i>Admin option 1.2: Defining Index Fields.....</i>	<i>12</i>
<i>Display Format Defaults.....</i>	<i>15</i>
<i>Admin option 3: Browse OMCS/RCX audit log.....</i>	<i>17</i>
<i>Admin option 4: Maintain OMCS/RCX Databases</i>	<i>17</i>
RENAMING THE RCX DATABASES	18
<i>Find the OMCS/RCX ("Indexing") databases</i>	<i>18</i>
<i>Identify the datasets to be renamed.....</i>	<i>19</i>
<i>Rename the datasets:</i>	<i>20</i>
<i>Make the Control database useable:</i>	<i>20</i>
<i>Make the new databases known to OMCS/RCX :.....</i>	<i>22</i>
<i>Make the Index and Work databases useable:</i>	<i>22</i>
<i>Make batch JCL and ISPF panel changes</i>	<i>22</i>
REPORT EXTRACT SCRIPTS	25
INTRODUCTION TO EXTRACT SCRIPTS.....	25
<i>Introductory Script Example.....</i>	<i>25</i>
SCRIPTING LANGUAGE REFERENCE	31
<i>General syntax rules:.....</i>	<i>31</i>
<i>Statement Summary by Verb:</i>	<i>31</i>
EXTRACT.....	31
FOR <entity> FROM <expr> TO <expr> [BY n]	31
IF.....	31
LEAVE.....	31
MAP	31
NEXT	31
OPTIONS	32
<i>Statement Reference by Verb:</i>	<i>32</i>
EXTRACT indexfieldname[=mapfieldname valuestring]	32
FOR <PAGE LINE POS> FROM <expr> TO <expr> [BY n].....	32
IF.....	32
LEAVE.....	32
MAP	32
NEXT	34
OPTIONS option1 [option2 ..]	34
The OPTIONS statement sets and resets execution-time options.	34

THE INDEX QUERY APPLICATION	35
INTRODUCTION	35
THE IQA DISPLAYS	35
<i>The IQA selection criteria display:</i>	35
<i>The IQA selected transactions list display:</i>	35
OMCS/RCX UTILITY PROGRAMS.....	37
OMCXEXTR	37
<i>Sample JCL for OMCXEXTR</i>	37
<i>OMCXEXTR parameters</i>	37
<i>Scheduling execution of OMCXEXTR by OMCXEXJS</i>	37
<i>Example of OMCXEXTR Control Listing</i>	37
OMCXEXTR Messages	38
OMCX201e ddname aa not allocated	38
OMCX202e bad program parameters, rc=nn	38
OMCX203i processing file: aa bb with script cc	38
OMCX204i processing extract record set aa	38
OMCX205i run parms: aa	38
OMCX206e Unrecognised term: "aa"	38
OMCX207e scriptname not specified, required	38
OMCX209i yyyyymmdd-hhmmss START SCRIPT aa extract# bb	38
OMCX210e load failed for module aa	38
OMCX211I eof on i/p file after aa pages bb lines - rc=cc	39
OMCX212e file aa bb - no ASPLEVL and file not empty	39
OMCX213e missing extract audit record ("X" rec)	39
OMCX214e file's update history shows more "E" records than the file contains, possible seq. error	39
OMCX215I the program deleted %1 unaudited "E" records	39
OMCX216I skipping extract rec range aa already processed	39
OMCX217I file TRANDATE REGISTER last entry aa later than new entry bb, dump follows	39
OMCX218I file TRANDATE REGISTER structural error, dump follows	40
OMCX219I compile failed for script aa, rc=bb	40
OMCX223E There is no aa database defined in the settings register	40
OMCX224E DBSTART failed for the aa database "bb"	40
OMCX225w extract bypassed, no SDS file specified, rc=aa	40
OMCX226w extract bypassed, script aa not compiled, rc=bb	40
OMCX227i Script execution took aa seconds CPU, rc=bb, cc	40
OMCX228w extract# aa already present in extracts register	40
OMCX229e extract register record was missing, recreated	40
OMCX230e SIMULATE is in effect, wont create an EXTRACT# file	41
OMCX231w extract ended rc=aa, bb	41
OMCX232i started RCX aa database bb	41
OMCXHSKP	42
<i>Sample JCL for OMCXHSKP</i>	42
<i>OMCXHSKP parameters</i>	42
<i>OMCXHSKP control statement format and syntax rules</i>	42
<i>OMCXHSKP Control Listing</i>	43
OMCXHSKP functions: MAXCOND	44
OMCXHSKP functions: EXTRACT	44
OMCXHSKP functions: DIAGNOSE	46
OMCXHSKP functions: COMPRESS	50
OMCXHSKP control statement example	50
OMCXTDXU	53
<i>Sample JCL for OMCXTDXU</i>	53
<i>OMCXTDXU parameters</i>	53
<i>Example of OMCXTDXU Control Listing</i>	54

<i>OMCSTDxu Messages</i>	54
OMCX301e ddname <i>aa</i> not allocated.....	54
OMCX302e bad program parameters, rc= <i>aa</i>	54
OMCX303i processing file: <i>aa bb</i> , ASPLEVL rec follows:.....	54
OMCX304e omcxusr.init failed, rc= <i>aa</i>	54
OMCX305i run parms: <i>aa</i>	54
OMCX306e Unrecognised term: " <i>aa</i> "	54
OMCX307e extract# not specified, or invalid.....	54
OMCX308i <i>aa</i> ended, rc= <i>bb cc</i>	55
OMCX309e missing D record, no extracted fields attributes.....	55
OMCX310e sort failed, terminating.	55
OMCX311e reached eof on input extract file.....	55
OMCX312e error, cannot update compressed file <i>aa bb</i>	55
OMCX313E build resident EXTS structures failed, rc= <i>aa</i>	55
OMCX314I there are no more unprocessed extract files	55
OMCX315I file <i>aa bb</i> will be updated.....	55
OMCX316e locate error, date: <i>aa</i> time: <i>bb</i> E rec#: <i>cc</i>	55
OMCX317e insert error, E rec date: <i>aa</i> time: <i>bb</i> rec#: <i>cc</i>	56
OMCX318i inserting R rec for unaudited E recs <i>aa</i>	56
OMCX319e update-record failed, rec#= <i>aa</i> ndrec#= <i>bb</i>	56
OMCX320e OPEN failed for index file during Q rec update	56
OMCX321e LOCATE failed during Q rec update	56
OMCX322e UPDR failed during Q rec update	56
OMCX323e IRAC failed during Q rec update	56
OMCX324e CLOSE failed during Q rec update	56
OMCX325e unknown rc <i>nn</i> from Q rec update	56
OMCX326e unspecified error sorting keys for Q rec update	57
OMCX327i memory area... size.....	57
OMCX328I extract# <i>aa</i> not found in extracts register file	57
OMCX329E extract process rc not 00, 04, update bypassed for <i>aa</i>	57
OMCX330E extract <i>aa</i> is held, cannot process further	57
OMCX331i error diagnostics, save for support.....	57
OMCX332E extract <i>aa</i> duplicate for file <i>bb cc</i>	57
OMCX333i there are no eligible extracts to process.....	57
OMCX334E MAXTIME <i>aa</i> not decimal 0:1439	57
OMCX335E MAXTIME limit prevents further processing	58
OMCX336E EXTRACTS REGISTER error, cannot find <i>aa</i> entry	58
OMCX398i program %1 terminating with COND CODE %2 %3.....	58
OMCX399e open failed for file <i>aa bb</i> rc <i>cc</i>	58
THE INDEX DATABASE API	59
API INTRODUCTION.....	59
<i>The basic API functions are:</i>	59
<i>Outline of a typical API application:</i>	59
API CALLS REFERENCE.....	60
<i>Starting a session with an index database</i>	60
<i>Selecting items that match criteria</i>	61
<i>Discarding a set of selected items</i>	64
<i>Ending a session with an index database</i>	65
<i>Querying names & attributes of fields defined in an index database</i>	65

Introduction

OMCS Report Content Indexer (OMCS/RCX) provides a way to automatically “mine” existing report files, turning business information currently produced in the form of printed reports into high-performance query databases and conventional data feeds for other processes.

Exploit your enterprise's existing application reports resource

Your enterprise is likely to have substantial investment in existing applications that have been running reliably for years, faithfully generating reports crammed with valuable business information. While a pile of printed pages may long have ceased to be the most productive presentation format for this data, it's often awkward and expensive to reengineer the original application to try and meet today's requirements, let alone reliably predict tomorrows'.

If you have users who are currently searching paper reports by hand, or even viewing and searching flat files electronically, then RCX can be used to deliver criteria-based interactive querying immediately. Whether interim or long-term, RCX-based solutions are quick, low-cost and reliable.

Pressure to re-engineer applications, delivery mechanisms and presentation formats to meet evolving business needs never really goes away. Professionals need ways of responding quickly, within budget, with tailored, effective solutions. RCX provides easy-to-use middleware to rapidly mate important existing data resources to new information needs.

RCX Components:

Transaction Data Extractor (TDE) - a script-driven 'report scraper' and index builder.

Reports captured by OMCS are analyzed under the control of a user-written Extraction Script. An Index Database is built from the extracted data.

Interactive Query Application (IQA) - gives end-users immediate access to data.

The IQA requires no additional configuration, it's just ready to use against any Index Database. Key fields and returnable data fields are automatically defined. End-users can further control which data fields they see, sort responses on any field and choose a variety of print options..

Application Programming Interface (API) - middleware for mating to other processes.

The API returns transactions consisting of just those fields that the programmer has selected. The programmer may select data in memory, flat file or OMCS database file response modes. Individual transaction records may have fixed length fields, comma-separated values (CSV) or other formats to suit application needs.

Pre-requisite software

OMCS/RCX requires the OMCS base product, FMID ASEB200, at the OMCS 2.10 level or higher.

RCX Setup

Database requirements and sizing

Four databases are required

Control database: Used for definition and control files and audit logs. Use 10 cylinders.

Index database: Used to hold the index data itself. Assuming that that all of the detail data within the report is to be extracted and held within the index you should allow 1.4 times the original report size multiplied by the number of versions of the report that the index is to contain. The OMCS DDR facility allows secondary databases to be added at any time, so you may choose to allocate a smaller amount initially and add one or more secondary databases as more data is collected. *It is critically important that the index database does not run out of space during an index update process.*

Work database: Used to hold extracted data used for index building. Assuming that all of the detail data within the report is to be extracted you should allow approximately the same amount of space required for the report itself multiplied by the number of versions of the report to be processed in, say, a 72 hour period.

Reports database: Used by OMCS to stage captured report files until they have been analyzed by the Transaction Data Extractor (TDE) component. Once TDE processing is complete the report files may be archived and deleted. A typical TDE extraction will be scheduled as a batch job by an OMCS system task ("OMCS Writer") on the same day that a report is captured. Once archived the report file can be deleted from this database.

Database naming and usage notes

Naming conventions: When choosing dataset names for these databases may we suggest the following convention be applied: Let the second last dsname segment contain the id of the original report that's being indexed, or a valid abbreviation of it. Let the final segment be, as appropriate, **CTLDB**, **NDXDB**, **WRKDB** or **REPDB**.

Using one database for multiple roles: RCX can operate using the same physical database for more than one of these functional roles. For testing purposes, for indexes where relatively small quantities of data are being maintained or just to simplify data management issues, it may be desirable to use a single database for several functions. For situations where large quantities of index data (say > 3GB) are to be maintained then ASE strongly recommends that the Control, Index and Work database roles are assigned to separate datasets.

Using an existing Reports database: In many cases the reports to be indexed will already be being captured and held in an existing OMCS Version 2 database. You can use such a database as the Reports database for RCX. Alternatively, reports may be restored from an archive created by OMCS Version 2 or obtained from other sources, in which cases a newly allocated RCX database should be used.

Compatibility with OMCS V2: OMCS/RCX databases cannot be processed by OMCS Version 2 programs, however OMCS/RCX can read and write OMCS Version 2 databases.

Block size choice: ASE recommends that all RCX databases use half-track block sizes, for example 27998 for a 3390 device type.

Allocating the databases

Use ISPF option 3.2 *or* tailor and use sample JCL shown below to allocate the database:

Using ISPF 3.2

```

Organization . . . : PS
Record format . . . : F
Record length . . . : 27998
Block size . . . . : 27998
1st extent cylinders: ?      <-read the sizing notes
Secondary cylinders : 0      <-note zero secondary space!

```

Using the sample allocation JCL:

```

/* REFER: ASEN100.ASECNTL(RUNALOC)
/* Sample JCL for allocating one index database set
/* Alternatively you may use ISPF 3.2
/*-----//BR14      EXEC
PGM=IEFBR14
/*
//CTLDB      DD DSN=your.hlg.reportid.CTLDB,SPACE=(CYL,10),
//            DISP=(NEW,CATLG),DCB=(RECFM=F,BLKSIZE=27998,DSORG=PS),
//            UNIT=...,VOL=SER=...,...as required by your storage mgmt
/*
//NDXDB      DD DSN=your.hlg.reportid.NDXDB,SPACE=(CYL,????),
//            DISP=(NEW,CATLG),DCB=(RECFM=F,BLKSIZE=27998,DSORG=PS),
//            UNIT=...,VOL=SER=...,...as required by your storage mgmt
/*
//WRKDB      DD DSN=your.hlg.reportid.WRKDB,SPACE=(CYL,????),
//            DISP=(NEW,CATLG),DCB=(RECFM=F,BLKSIZE=27998,DSORG=PS),
//            UNIT=...,VOL=SER=...,...as required by your storage mgmt
/*
//REPDB      DD DSN=your.hlg.reportid.REPDB,SPACE=(CYL,????),
//            DISP=(NEW,CATLG),DCB=(RECFM=F,BLKSIZE=27998,DSORG=PS),
//            UNIT=...,VOL=SER=...,...as required by your storage mgmt
//

```

Note 1. OMCS/RCX databases are not formatted before use as OMCS Version 2 databases are. Database formatting takes place as required during normal processing.

Note 2. A single RCX database is restricted to a maximum of 4369 cylinders, even though the device type may have more than that number of cylinders available. If the Index database must be larger than 4369 cylinders you must use the OMCS DDR facility to allocate and start as many secondary databases as are necessary to achieve the size required. RCX allows up to 127 secondary databases to be allocated. See .ASECNTL(RUNDDR) sample JCL.

Setting Global RCX parameters

Invoking the RCX Administration ISPF option.

Choose the appropriate ISPF option to invoke the RCX Administration function for the your new control database. If this has not yet been done, please refer to "Appendix a – ISPF Installation".

When the RCX Admin application is invoked you will see the following display:

```

OMCS/RCX Main Menu
Command ==> 2_

 1 Maintain OMCS/RCX Application Controls
 2 Initialise Application Control Files
 3 Browse OMCS/RCX audit log
 4 Manage OMCS/RCX Databases

```

Admin option 2: Initialise Application Control Files

Type “2” and press ENTER. You should see the following audit display:

```

OMCS View OMCXINIT RESULTS
Command ==>

                                     Line 0 of..
***** Top of Data *****
Results of OMCX control file initialization:
-----

file: omcxaudt logfile
return code: 0
message: OMCX209I AUDIT initialisation successful

file: settings register
return code: 0
message: OMCX209I SETTING initialisation successful

file: fields register
return code: 0
message: OMCX209I FIELD initialisation successful

file: trandate register
return code: 0
message: OMCX209I TRDR initialisation successful

-----
Press the END key to return to the previous menu.

```

Press END to return to the admin menu.

```

OMCS/RCX Administration Main Menu
Command ==> 1_

 1 Maintain OMCS/RCX Application Controls
 2 Initialise Application Control Files
 3 Browse OMCS/RCX audit log
 4 Manage OMCS/RCX Databases

```

Admin option 1: Maintain OMCS/RCX Application Controls

Type “1” and press ENTER. You should see the following:

```

OMCS/RCX Application Controls
Command ==> 1_
    1 Global Settings
    2 Field Definitions

```

Admin option 1.1: Global Settings

Type “1” and press ENTER. You should see the following:

```

OMCS/RCX                      settings register
Command ==>                      Scroll ==> PAGE
                                   Line 1      of 6
Line cmds:  D delete  I insert  S view  U update
lc setting----- value----- last-mod
.. dsn_control_files_database      ASE.ASEN100.DBASE      MYLOGON
.. dsn_extract_scripts_library      MYLOGON
.. dsn_index_files_database         MYLOGON
.. dsn_jcl_skeletons_library        MYLOGON
.. dsn_report_files_database        MYLOGON
.. dsn_work_files_database          MYLOGON

```

Note that the *dsn_control_files_database* entry already has a value.

Type a “u” line command against the setting entry for *dsn_index_files_database* and press ENTER. You should see the following:

```

OMCS/RCX                      Global Settings dsn_index_files_database
Command ==>
variable name . . dsn_index_files_database
value . . . . .

```

Type or paste the full dataset name (without apostrophes) into the value field and press ENTER.

Repeat for the *dsn_report_files_database* and *dsn_work_files_database* entries.

Leave the *dsn_extract_scripts_library* and *dsn_jcl_skeletons_library* entries blank at this stage. Press END to return to the Application Controls menu:

```

OMCS/RCX Application Controls
Command ==> 2_
    1 Global Settings
    2 Field Definitions

```

Admin option 1.2: Defining Index Fields

When RCX extracts data from a report line or page it assigns the data to some Extract Field. The Transaction Data Extract process validates and transforms extracted report data according to the attributes of the extract field that the data is assigned to.

BEFORE DEFINING INDEX DATA FIELDS

Field definitions need to be “robust” because there are costs involved in making changes to them once index building has commenced. Some changes cannot be made without affecting processing integrity, others may be trivial to make but might cause confusion amongst your user population. So, to begin with, you should have a clear idea of how the extracted data is going to be perceived and used. Remember that the field names chosen will be seen by users of the IQA. Gaining a high level of

familiarity with the source report(s) before starting the field definitions may cause your ideas about field names to change as well as giving you a better understanding of the range and type of data that can be present. Once these points are reasonably clear, then you can start defining fields.

Try to standardize field names within an application, a business area or even across the complete enterprise. Different people may choose different names for the same data item, perhaps because two different source reports used slightly different field heading titles for the same data element. For example: CARDNUMBER and CARDHOLDER. It's probable that these two fields contain exactly the same data type and it may help to reduce confusion and uncertainty on the part of both business users and IT technicians if the organization chooses one name as a standard.

USING THE FIELD DEFINITION DATA ENTRY SCREENS

```

OMCS/RX Application Controls
Command ==> 2_
    1 Global Settings
    2 Field Definitions
  
```

Type "2" and press ENTER. You should see the following:

```

OMCS/RX                      field definition
Command ==>                      Scroll ==> PAGE
                                   Line 1      of 2
Line cmds:  D delete  I insert  S view  U update
lc field name----- fl#  ksz description----- last-modified-----
.. DATE                      1    0 Transaction date field  MYLOGON  20030603 112939
.. TIME                      2    0 Transaction time field  MYLOGON  20030603 112939
  
```

RCX reserves two field names, DATE and TIME, which it insists must always be defined. They are assigned field numbers (fl# heading) 1 and 2. These entries cannot be updated.

Type ADD in the primary command field, or "i" in any line command field, and press ENTER to commence the definition of a new field. You should see the following:

```

OMCS/RX                      field definition new
Command ==>
ASEOB183 create a new FIELD
field number . : 003
fieldname . . . (full name for field)
description . .
key size . . . (1-255 bytes)
remove spaces . (Y to remove embedded spaces from values)
translate char . replace these..
.. to char . ..with these
Storage format .
Display format .

Help text for this field:
-----
Your administrator has not defined any help text for this field yet.
-----
  
```

field number RCX automatically assigns a unique number to each field.

- Fieldname** choose a field name of up to 16 characters. Use any combination of the letters a-z, digits 0-9, @, #, \$, . The name you choose must be unique within the index. Names should be meaningful or descriptive of the field content, like COST_AMOUNT or ACCT_NUM. Try to avoid having a field name that is, say, twice the length of the field's data values. (A useful approach is to derive the index field names from the field headings in the source report).
- description** A brief description of the field. This could include, for example, report name and field heading for the source of this field's data.
- key size** If this field is not to be a key field for selecting index data then key size must be left blank or set to zero.
- If this field is to be a key field for selecting index data then key size must be the same as the length specified in the *storage format* field. Remember that a storage format of N11, for example, will hold up to 22 decimal digits. (you would specify 11 for key size).
- remove spaces** If source values may contain embedded spaces for reasons that don't change the field's value, such as readability of long numbers, then setting this attribute to "Y" will cause RCX to remove embedded spaces before storing the value.
- translate char** When any of the characters typed here are found in this field they will be replaced with the corresponding character in the **.. to char** field.
- .. to char** see description of **translate char** above.
- storage format** sometimes referred to as **held-as**, this attribute describes how the extracted data is to be held in the index database. It consists of a type code letter and an optional byte count. Supported type codes are:
- Cn** data is held as a fixed-length character string of n bytes, padded on the right with spaces (blanks).
- Nn** data is held as a fixed-length string of n bytes with two decimal digits per byte. The maximum number of digits that can be held in an N type field is 2*n. N format requires that the source data consist solely of decimal digits. Because N type fields offer an effective two to one compression ratio over C type fields they should be used wherever possible.
- V** data is held as a varying length character string with a length prefix byte. V type fields also incorporate run-length encoding for contiguous blanks within the data. Use V type for long fields with unpredictable content like titles, descriptions, address lines and fields that may be substantially blank.
- \$n** data is held as a binary count of "cents". Used for decimal currency fields where the amount is present as a string of digits, the rightmost two of which represent cents. Any optional markup or "furniture" characters, such as periods, commas, "\$" or other currency symbols will be ignored with only the decimal digits being picked up. The data is stored as a n-byte binary count of cents where n may be 1, 2, 3 or 4. These lengths correspond with maximum values of \$2.55, \$327.67, \$167,772.15 and \$21,474,836.48 respectively. Choose n such that it will accommodate the largest value likely to be found in the source data.

T data is held as a three-byte string with two decimal digits per byte. The source data is assumed to have the general format hh.mm.ss where hh=hours, mm=minutes and ss=seconds and “..” represents any optional markup or “furniture” characters, such as “-“ or “.”.

D data is held as a four-byte string with two decimal digits per byte. The source data is assumed to have the general format yyyy.mm.dd –or- yy.mm.dd where yy[yy] is a year number, mm a month number, dd a day number within the month and “..” represents any optional markup or “furniture” characters, such as “/ or “-“or “.”. with two decimal digits per byte. When the source data is exactly six bytes long the yymmdd format is assumed and the value 2000 is added to the yy component to convert it to yyyy format. In all other cases the year is assumed to be four digits. Date fields often require the use of REBUILD AS (see MAP statement description).

display format Please leave this field blank at this time.
See notes under Display Format Defaults below.

Help text for this field You may enter up to 10 lines of descriptive text for this field. This information will be displayed when an IQA user hits the HELP key on the selection criteria display with the cursor is in this field.

Press ENTER at any time to have current data validated.

Note that you cannot define more than 255 fields for a single index.

Press END to save the field definition and return to the field selection list.

When the field definitions are done you may move on to creating a Report Extraction Script.

Display Format Defaults

By default RCX will present data retrieved from the index database in character string form. Depending upon the *storage format* type code in the appropriate field definition, data retrieved from the index may differ in some essentially cosmetic ways from the original source data. The following list describes possible differences:

All formats: RCX Transaction Data Extraction (TDE) may skip leading spaces as it attempts to find non-blank data within the source “window” defined by the MAP statement within a particular extraction script. Excess data may also be truncated.

C embedded spaces in the original source data may have been elided.

N embedded spaces in the original source data may have been elided.

D a variety of source date formats always result in yyymmdd responses.

T a variety of source time formats always result in hhmmss responses.

\$ a variety of source date formats always result in \$.nn.cc responses.

V no other changes

RCX definitions are now complete.

Admin option 3: Browse OMCS/RCX audit log

Type “3” and press ENTER. You should see the following:

```

OMCS/RCX Audit Log
Command ==>
Scroll ==> PAGE
Line 1 of nnnn
hhmmss job/user event----- filename filetype related-information-----
ASPLeVL=0001
DAY day dd mth yyyy ddd -----
150812 userid init SETTING settings register OMCX209I SETTING initialisat
150812 userid init FIELD fields register OMCX207 ..
150813 userid init TRDR trandate register OMCX209 ..
..

```

This scrollable display shows recent OMCS/RCX events for this control database. From time to time ASE may request that you use the PRT command to extract ranges of lines from this audit log for diagnostic purposes. Use the HELP key to get information about available primary commands. Use the EXIT key to return to the OMCS/RCX Administration Main Menu.

Admin option 4: Maintain OMCS/RCX Databases

Type “4” and press ENTER. You should see the following:

```

OMCS/RCX Manage Databases
Command ==>

1 Control database - 'your.control.database'
2 Index database - 'your.index.database'
3 Reports database - 'your.reports.database'
4 Work database - 'your.work.database'

```

Select the control database by typing “1” and pressing ENTER. You should see the following:

```

OMCS/RCX Control database - 'your.control.database'
Command ==>

1 database information
2 directory display
3 Access Control Facility rules
4 Archive Management Facility (AMF)
5 Dynamic Database Reconfiguration (DDR)

```

Each of these options will take you to a corresponding display which should be familiar to a person experienced in general OMCS administration. Remember to press the HELP key whenever you need information about available primary or line commands on any display.

Access the same five options for each of the OMCS/RCX databases by choosing the corresponding option on the “Manage Databases” menu.

Renaming the RCX databases

This section deals with a requirement that may occur during the life of the OMCS/RCX application. It is not part of setting up OMCS/RCX.

The OMCS/RCX application uses a set of four primary databases. Only the Control database is referenced via JCL. The remaining databases are located via a configuration file located in the Control database. Some of the databases may use the OMCS Dynamic Database Reconfiguration (DDR) feature to increase their size by the addition of secondary databases. The combination of the configuration file and DDR means that the renaming process is not trivial. Successful renaming simply requires that the following notes be carefully followed.

The process is described as seven steps, some simple, some a little more involved. Let us apologize in advance if these notes repeat things that you already know very well, however we cannot assume too much about the reader's level of experience and thus must err on the side of "too much information".

The stages are:

- Find the OMCS/RCX ("Indexing") databases
- Identify the datasets to be renamed.
- Rename the datasets:
- Make the Control database useable:
- Make the new databases known to OMCS/RCX :
- Make the Index and Work databases useable:
- Make batch JCL and ISPF panel changes

Note: Please do just what's asked and in the order requested.

Find the OMCS/RCX ("Indexing") databases

The OMCS/RCX application uses a set of four primary databases:

- Control database
- Index database
- Work database
- Reports database (technically not an OMCS/RCX database, just referenced during extracts)

In the JCL for each of the three batch utilities:

- the extract program: `//.. PGM=ECLX, PARM='OMCXEXTR ..`
- the index update program: `//.. PGM=ECLX, PARM='OMCXTDXU ..`
- the housekeeping program: `//.. PGM=ECLX, PARM='OMCXHSKP ..`

..find the //OMCXCTL DD statement. This gives the dsname of the control database.

Note: while the dsname in the //OMCXCTL DD statement in each of these job steps will eventually need changing, DON'T DO THIS YET! We want to do these JCL changes last, so that these batch jobs cannot access any of the OMCS/RCX databases until the entire renaming has been completed.

To determine the names of the Index and Work databases:

- Invoke the OMCS/RCX Administration ISPF application;
- choose option 1 Maintain OMCS/RCX Application Controls
- Choose option 1 Global Settings
- The settings register display looks like:

```

OMCS/RCX                                     settings register
Command ==>

Line cmds:  D delete  I insert  S view  U update
lc setting----- value-----
.. database_for_control_files      your.control.database
.. database_for_index_files        your.index.database
.. database_for_t_reporfiles       your.report.database
.. database_for_work_files         your.work.database
.. library_for_extract_scripts

```

You can see the dsname of the Control, Index, Report and Work databases under the field headed "value-----". You can also use the "s" line command to see the value displayed on a separate panel.

Identify the datasets to be renamed

In addition to the primary database dsnames, you must identify any secondary databases that have been defined to extend the primaries.

To determine what those secondary database dsnames are :

- Invoke the OMCS/RCX Administration ISPF application;
- choose option 4 Manage OMCS/RCX Databases
- choose option 1, Control database

choose option 5 Dynamic Database Reconfiguration (DDR)

Record the dsnames of any secondary databases listed.

- choose option 2, Index database

choose option 5 Dynamic Database Reconfiguration (DDR)

Record the dsnames of any secondary databases listed.

- choose option 3, Work database

choose option 5 Dynamic Database Reconfiguration (DDR)

Record the dsnames of any secondary databases listed.

- Note: Once you start renaming datasets you won't be able to re-run these administrative options to determine the dsnames, so it's important to make a clear record of them all before commencing the rename process.

Rename the datasets:

Once you know the dsnames of all the datasets you may commence the renaming. You can use IDCAMS to do that.

Rename the control database primary dataset first. This will ensure that:

- No OMCS/RCX jobs are running at the moment (RENAME cannot acquire control of the dname if it's allocated to any job or TSO user);
- Any jobs inadvertently submitted after you rename the control database primary dataset won't be able to start, because they won't be able to allocate the control database using its old name.

Make the Control database useable:

The ISPF panels that provide access to the OMCS/RCX applications will contain the dname of the control database.

- Change **ONLY** the ISPF panel that invokes the OMCS/RCX Administration application by replacing the old control database dname with the new dname.
- To prevent users from inadvertently invoking the Interactive Query Application (IQA) before you have completed the rest of the administrative detail described below, DO NOT change the ISPF panel that gives access to that application yet.

The control database may have secondary databases. The primary database contains the names of any secondary databases and when you have renamed databases these entries must be changed.

- Invoke the OMCS/RCX Administration application;

At this point you may see some line-mode DAIO.. messages as the database access method automatically adjusts the control database's self-describing entry for the primary database (which will now have a different name). If you don't see any such messages that's ok, it either means there are no secondary databases –or– that the self-describing entry for the primary has already been adjusted. If line mode messages do appear then when you see the three asterisks (***) just press ENTER to see the initial screen for the Administration application.

When the initial screen for the Administration application displays, it should have a highlighted message box at the bottom of the screen containing a message about the application automatically adjusting the control database entry in its OMCS/RCX configuration file:

OMCX220W the control database has been renamed from 'oldname' to 'newname'
the self-describing settings register entry has been adjusted accordingly

If this message does not appear it may simply indicate that this is not the first time that the Administration application has been run since the control database was renamed.

- For the control database only, run Dynamic Database Reconfiguration (DDR) as follows.

Invoke the OMCS/RCX Administration application:

choose option 4 Manage OMCS/RCX Databases

choose option 1, Control database.

choose option 5 Dynamic Database Reconfiguration (DDR)

DDR will display a list of datasets, primary first, over which the database is mapped. The display will look like:

```

                                Dynamic Database Reconfiguration List -
Command ==>

Line cmds:  A Alloc  C Change  F Format  S Start  P Stop  D Delete  I info
lc No.  Status   Dataset Name                               Volume Alloc
.. 000 PRIMARY   your.control.database                         ASE005 CYL
..

```

- If the database has secondary databases defined, then you must use the "c" (change) line command to update the dsname data for each secondary.

The "c" (change) line command will cause a display like:

```

                                DDR Change Secondary Database
Command ==>

Please enter a new value for the secondary database
Database number : 001
Database name . . 'your.primary.database.name'

This database is already allocated. Only the

Use the EXIT command to process, or CAN to cancel

```

Over-type the old dataset name with the new and press the EXIT key (described as the "EXIT command" on the screen) to save the updated entry.

When all entries have the correct dataset names, press the EXIT key to terminate DDR.

- Verify that the rename of the control database was successful

At this point all the "Manage Databases" functions should be available for the Control database. Choose option 1 database information and ensure that it presents the "Cluster Information" display. A failure here means the names in the DDR datasets list are wrong.

Press the EXIT key several times to terminate OMCS/RCX Administration.

Warning: DO NOT attempt to repeat the above process for the Index and Work databases until you have done the next step!

Make the new databases known to OMCS/RCX :

OMCS/RCX holds the names of the Index, Work and Reports databases in a configuration file in the control database.

Invoke the OMCS/RCX Administration ISPF application;
 choose option 1 Maintain OMCS/RCX Application Controls
 choose option 1 Global Settings
 The settings register display looks like:

OMCS/RCX		settings register
Command ==>>		
Line cmds:	D delete	I insert S view U update
lc setting-----	value-----	
.. database_for_control_files	your.control.database	
.. database_for_index_files	your.index.database	
.. database_for_t_reporfiles	your.report.database	
.. database_for_work_files	your.work.database	
.. library_for_extract_scripts		

- Use the "u" line command against the entry "database_for_control_files" and over-type the dsname on the resulting display with the new Control database dsname. Press the EXIT key to save the change.
- Use the "u" line command against the entry "database_for_index_files" and over-type the dsname on the resulting display with the new Index database dsname. Press the EXIT key to save the change.
- Use the "u" line command against the entry "database_for_work_files" and over-type the dsname on the resulting display with the new Work database dsname. Press the EXIT key to save the change.
- Press the EXIT key several times to return to the Administration Main Menu.

Make the Index and Work databases useable:

It is possible that the Index and Work databases have secondary databases defined.

- For the Index database, choose option 5 Dynamic Database Reconfiguration (DDR) and change the dsnames for any secondary databases to the correct names.
- For the Work database, choose option 5 Dynamic Database Reconfiguration (DDR) and change the dsnames for any secondary databases to the correct names.

Make batch JCL and ISPF panel changes

- For each of the extract, index update and housekeeping jobs, change the dsname in the //OMCXCTL DD statement to point to the control database.
- Change the ISPF panel that invokes the Interactive Query Application (IQA) so that it names the control database in the parm field.

The renaming of the OMCS/RCX databases is now complete.

Report Extract Scripts

The Transaction Data Extractor component of RCX requires an Extract Script to tell it how to extract transaction data from a source report or data file

Introduction to extract scripts

Scripts are typically created using ISPF edit or any suitable text editor. They are stored as library members and are executed as a step in a batch job. The results are a control listing and an extract file in the RCX “work” database.

Scripts are “compiled” at run time. If the compile process discovers errors it will not attempt to run the script against the source data. RCX maintains an audit trail of script executions.

Introductory Script Example

[Note, this example is not the reference for the scripting language, it is intended to introduce the reader to the scripting language and its features]

Imagine that we wanted to extract some details from the following report and create a high-speed lookup database using RCX.

```

1AUTOFAST CREDIT INC.
0AFCR023                                CREDIT CARD REQUEST/CHANGE
                                         -----
0CREDIT CARD DATABASE                   REGION: 0001 NEVADA
                                         -----
      DATE      TIME      USER  ACTION      EMBOSSED      CARD CARD  CARD
      DATE      TIME      USER  ACTION      CARD NO.      STAT TYPE EXPIRY
      -----
0 6/08/90  14:24:42  CPJS  ADD      NEW:  7052 85422698 02200
0 6/08/90  14:25:45  CPJS  ADD      NEW:  7052 85422698 02218
0 6/08/90  14:26:12  CPJS  ADD      NEW:  7052 85422698 02226
0 6/08/90  15:00:15  CPJS  ADD      NEW:  7052 21978346 03603
1AUTOFAST CREDIT INC.
0AFCR023                                CREDIT CARD REQUEST/CHANGE
                                         -----
0CREDIT CARD DATABASE                   REGION: 0002 WASHINGTON
                                         -----
      DATE      TIME      USER  ACTION      EMBOSSED      CARD CARD  CARD
      DATE      TIME      USER  ACTION      CARD NO.      STAT TYPE EXPIRY
      -----
0 6/08/90  16:12:02  CPJS  ADD      NEW:  7052 85609154 00062

```

After studying the report and discussing with business users what they need from it we decided to define the following index fields (see “[Admin option 1.2 – Defining Index Fields](#)” described in previous pages):

```

field name----- fl# ksz description-----
ACTION              6   0 action taken
CARD_NO             7  10 credit card number
DATE                1   0 Transaction date field
REGION_NAME         9   0 credit card region name
REGN#               8   2 credit card region number
TIME                2   0 Transaction time field
USER                5   5 user id field

```

Three of the fields (card_no , regn# and user) are defined as key fields.

The following script will extract the data we want from the Credit Report. Note that each statement verb has been shown **bold** for clarity:

```
options    acceptrc=4 +      /* continue while internal rc < 5
            showflds +      /* show field definitions on report
            trace +          /* turn detailed extract tracing on
            verify           /* print lines to verify file position

for page 1 onward      /*----- find first report page in listing
    if line 1 for 1 col 1 on begins ("1AUTOFAST CREDIT")
        log
        leave

for page * onward      /*----- process credit report pages

    if line 1 for 1 col 1 on not begin ("1AUTOFAST CREDIT")
        leave

    map line 4 col 62 on +    /* map fields in 4th heading line
        regn#(c4) +          /* nnnn
        (1) region_name(c32) /* state or region name

    for line 9 on by 1      /* process detail lines on current page

        if line * for 1 col 4 for 1 not contain ("/")
            next line      /* ..skip this line->

        map line * col 1 on + /* map fields in this line
            (1) date(c8,+      /* dd/mm/yy or d/mm/yy
                        TRANSLATE( ),TO(0),+
                        REBUILD(dd/mm/yy),AS(19%y%m%d)) +
            (2) time(c8) +     /* hh:mm:ss
            (2) user(c5) +     /* userid
            (2) action(c12) +  /* action
            (4) card_no(C19,remove_spaces) /* card number

        extract date=date +    /* get transaction date from t_date
            time=time +        /* get .. time .. t_time
            user +             /* extract remaining fields
            action +
            card_no +
            regn# +
            region_name
```

The following text takes each statement in turn and describes it's effect upon the extraction process.

options **acceptrc=4** +
showflds +
trace +
verify

During the extract, the highest allowable internal code will be 4, index field details are to be printed on the extract report, extract processing is to be traced and the current position within the report file is to be shown by printing record, page and line numbers and text from the line itself. The **showflds**, **trace** and **verify** options are typically used as diagnostic aids when setting up a new extract script.

for page 1 onward

The group of indented statements that follow are to be performed for each page in the report, starting with the first page. Note that the “scope” of this **for** statement extends to the line prior to the next **for** statement, ie. “**for page * onward**”.

if line 1 for 1 col 1 on begins (“1AUTOFAST CREDIT”)

log
leave

This group of statements skips any leading pages that are not AUTOFAST CREDIT report pages. It's not uncommon to find that the report you want to scan contains pages which do not contain the detail that you want to extract and index. Typical examples are banner pages or summary pages.

If starts at column 1 of the first line on the page and scans for the text “1AUTOFAST CREDIT”. If that condition is satisfied then the group of indented statements that follow will be executed, if not then they will be skipped.

log prints the image of the current line of the report on the extract report and **leave** terminates the repetition of the **for page 1 onward** statement group, leaving the file positioned to the page just inspected. If **leave** was not present then the extract would automatically position to the next page and repeat the indented statements. **Leave** causes control to pass to the next **for** statement, ie. “**for page * onward**”.

for page * onward

The group of indented statements that follow are to be performed for each page in the report, starting with the current page. Note that the first page examined here will be the last page examined by the previous “**for...**” logic. Note that the “scope” of this **for** statement extends to the last line of the script.

if line 1 for 1 col 1 on not begin (“1AUTOFAST CREDIT”)

Starting at col 1 of the first line on the page, this statement checks for the text “1AUTOFAST CREDIT”. If that condition is not satisfied then the indented **leave** statement that follows will be executed, terminating the “**for page * onward**” logic group. Note that this will terminate the script as the scope of this **for** statement extends to the end of the script.

```
map line 4 col 62 onward +
    regn#(c4) +
    region_name(c32)
```

This statement associates the field names `regn#` and `region_name` respectively with columns 62 to 65 and columns 67 to 98 of the 4th line on the current page. This mapping will remain valid only while this page is the current page.

for line 9 on by 1

The indented statements that follow this one are to be performed for each single line on the current page, starting with the 9th line. Note that by omitting the “page ..” clause from this `for` statement we imply the current page.

```
if line * for 1 col 4 for 1 not contain (“/”)
    next line
```

This pair of statements causes lines that don’t have “/” in column 4 to be ignored. The asterisk (*) in place of a starting line number is interpreted as meaning “the current line”. The **next line** statement causes the program to step to the next line on the page and re-perform the preceding **for line ..** statement.

```
map line * col 1 on +
    (1) date(c8,+
        TRANSLATE(,TO(0),+
        REBUILD(dd/mm/yy),AS(19%y%m%d)) +
    (2) time(c8) +          /* hh:mm:ss
    (2) user(c5) +          /* userid
    (2) action(c12) +       /* action
    (4) card_no(C19,remove_spaces) /* card number
```

This statement associates several field names with column ranges in the current line. The first field, `date`, occupies columns 2 to 9. In the Fields Register we defined `date` as type D, meaning it must be either a 6 or 8-digit field of format `yymmdd` or `yyyymmdd` respectively. In fact the extract process will tolerate embedded markup characters (like “/”, “-” or “:”) between the year, month and day number components of the source data for D type fields. However, the source data in this report does have problems that we must correct if a correct date is to be extracted.

First, the day number field has a leading space when the day number is less than 10. We compensate for this with `TRANSLATE(,TO(0))` so that “6/08/90” becomes “06/08/90”.

Second, the date components are not in the order that date extraction requires. To correct for this we provide a map of the source data showing where each component starts and how long it is by using `REBUILD(dd/mm/yy)` (noting that “yy” simply means the “y” field is two characters long) and then we show how we want the rebuilt data to look with `AS(19%y%m%d)`.

The final problem, that this two-digit date has “19” missing from the year component, was also fixed up by the `AS(...)` option where we put a “19” in front of whatever the value of %y is from the report line. Note that by default a two-digit year will be prefixed with “20” to make it a four-digit value. Usually this will be what’s required, but not always, as this example shows.

```
extract date=date +  
        time=time +  
        user      +  
        card_no   +  
        action    +  
        regn#     +  
        region_name
```

The extract statement pulls the data from the corresponding parts of each of the current line(s) (and in this case, two fields that always come from line 4 on the current page) and writes an extract transaction record that will later be used to update the index database for this report.

There are no more statements. The block of statements indented under “**for line 9 on by 1**” is executed until the end of the current page is reached. That causes the “**for page * onward**” statement to be re-executed until eventually there are either no more pages –or- a page is reached which does not have “1AUTOFAST CREDIT” at the start of the first line.

When the final line of the final page has been inspected, this script will end, making the extracted data available for the index update process to merge into the appropriate index database structure.

Scripting Language Reference

General syntax rules:

Statements begin with a **verb**, or **operation**, and must reside entirely within positions 1 to 72 of the script lines.

A “/*” pair on any line indicates that all positions to the right are comments. Comments are ignored.

If the last non-blank, non-comment character on any line is a plus sign (+) then this statement is continued from the first non-blank character on the next line.

Statements are generally case-insensitive.

The terms within a statement are generally separated from each other by one or more spaces.

Logic flow is determined by FOR and IF statements, each of which tests a proposition. If the proposition is true then the following indented statement group will be performed.

Statement Summary by Verb:

EXTRACT

The EXTRACT statement causes an extract transaction record to be written. The fields to be included in the record are named on the EXTRACT statement

FOR <entity> FROM <expr> TO <expr> [BY n]

The FOR statement indicates the start of a group of statements that is to be performed repeatedly. The following indented statement group is performed until the terminal condition specified by the operands of the FOR statement occurs or until a LEAVE statement is performed.

IF

IF marks the start of a group of statements to be performed once. If the proposition tested by the IF statement is satisfied then the following indented statement group will be performed.

LEAVE

LEAVE ends the execution of an indented group of statements being performed under the control of a FOR statement.

MAP

MAP statements define the position, length and possible transformations for source fields that may be named on an EXTRACT statement.

NEXT

NEXT skips the execution of the rest of the statements in an indented statement group under the control of a FOR statement and re-executes the FOR statement.

OPTIONS

The OPTIONS statement sets and resets execution-time options.

Statement Reference by Verb:

EXTRACT *indexfieldname*[=*mapfieldname*|*valuestring*] ...

The EXTRACT statement causes one extract transaction record to be written. The fields to be included in the record are named on the EXTRACT statement

- ***indexfieldname*** The name of an index field.
- ***mapfieldname*** The name of the MAP field that provides the data for this index field.
- When *mapfieldname* is the same as *indexfieldname*, then “=*mapfieldname*” may be omitted.
- ***valuestring*** A way of providing a specific value for this field.
- The first two *indexfieldnames* specified on any EXTRACT statement must be **DATE** and **TIME**.
- Note that when time is not relevant to the data being extracted, simply specifying TIME=000000 will provide a literal value.

FOR <PAGE|LINE|POS> FROM <expr> TO <expr> [BY *n*]

The FOR statement indicates the start of a group of statements that are to be performed repeatedly. The following indented statement group is performed until the terminal condition specified by the operands of the FOR statement occurs, or until a LEAVE statement is performed.

- <PAGE|LINE|POS> this required keyword term identifies what entity is being referred to by the for statement. The three possibilities are **page**, **line** and **pos** (position). **Pages** may be specified instead of page and **lines** may be specified instead of line.
- <expr> the starting and ending page, line or position can be specified as a number (1,2,...) or as a symbol (* means the current value) or as an expression like *+1.
- **BY *n*** specifies the increment *n* used for each iteration of FOR. If the BY clause is omitted then the increment *n* defaults to 1.

IF <relational proposition>

IF marks the start of a group of statements to be performed once. If the proposition tested by the IF statement is satisfied the following indented statement group is performed.

LEAVE

LEAVE ends the execution of an indented group of statements being performed under the control of a FOR statement.

MAP <line_expression> <cols_expression> <field_descriptor(s)> ...

Each MAP statement defines one or more source fields to be extracted (see the EXTRACT statement).from a specified source line.

<line_expression> LINE [FROM] *numexpr*

LINE is a required keyword;

FROM is an optional keyword which may be included for clarity.

numexpr identifies which line the fields are contained in.

n means "the *n*th line on the current page";

*** means "the current line" (ie. the line we are currently positioned at);

**+n* means "the *n*th line following the current line";

**-n* means "the *n*th line preceeding the current line".

<cols_expression> COL|POS [FROM] *numexpr* [TO|FOR *numexpr* | ON[WARD]]

COL is a required keyword, [POS is an accepted synonym for COL];

FROM is an optional keyword which may be included for clarity.

numexpr identifies which columns (or positions) in the line the fields are contained in.

n means "the *n*th column in the current line";

*** means "the current column" (ie. the column we are currently positioned at);

**+n* means "the *n*th column following the current column";

**-n* means "the *n*th column preceeding the current column".

TO/FOR/ON/ONWARD one of these alternative keywords may be specified as part of defining where the source area of the line ends. If none are specified then **ON** is assumed.

If **TO** or **FOR** is specified then they must be followed by *numexpr* to specify the rightmost column of the source area over which the named fields are mapped.

The **ON** and **ONWARD** keywords indicate that the source area extends to the end of the line.

<field_descriptor(s)> [(*n*)] *fieldname*(*tnnn*[,*transformlist*])

(*n*) specifies how many source data columns to skip. If omitted, (0) is assumed;

fieldname is the mapfieldname which may be referenced on an EXTRACT statement;

tnnn *t* is the typecode specifying the format of the source data:

C Character string data;

nnn the length in bytes of the source data field. Defaults to 1 if omitted

transformlist One or more optional terms defining manipulation or transformation of the source data one of these alternative keywords may be specified as part of defining where the source area of the line ends. If none are specified then **ON** is assumed.

If **TO** or **FOR** is specified then they must be followed by *numexpr* to specify the rightmost column of the source area over which the named fields are mapped.

The **ON** and **ONWARD** keywords indicate that the source area extends to the end of the line.

Example:

```
MAP LINE * FROM COL 6 FOR 17 + /* define a 17 character area
      BSB#(C7) + /* first 7 chars are the BSB number..
      (1) /* ..then skip one character..
```

```

ACCT(C9,+          /* then a 9-character account number
TRANSLATE(-),TO( ),+ /*
REMOVE_SPACES)

```

The map covers all or part of a single source line or of a group of adjacent source lines. ("mapfields") define the position, length and possible transformations for source fields that may be named on an EXTRACT statement.

NEXT

NEXT skips the execution of the rest of the statements in an indented statement group under the control of a FOR statement and re-executes the FOR statement.

OPTIONS option1 [option2 ..]

The OPTIONS statement sets and resets execution-time options.

Keyword options that may be specified are:

- **acceptrc=n** Internal return codes > **n** will terminate script execution. When **n** is > 3, messages about trivial errors will not be written.
- **showflds** The program prints a list of field names and attributes.
- **simulate** The script executes but does not write an extract detail file.
- **trace** The program traces individual field data extractions.
- **verify** Each time a new pagetop is reached the program will print the pagetop line
.

The Index Query Application

Introduction

The Index Query Application (IQA) runs as an ISPF dialog. The IQA retrieves data from transactions that satisfy the user's index key values and displays it in the form of a scrollable list. The user may view and/or print item(s) from the list and optionally sort the list on any field. The IQA is designed to allow the installation to add new primary and/or line commands to the basic application.

The IQA displays

The IQA selection criteria display:

```
Card Services Index - selection criteria
command ==>

type in any of the following fields and press ENTER to select transactions
ord fieldname----- argument(s)
  not-before          ==> 22aug00          eg: yymmdd or 25JUN orJun25
  not-after           ==>                .. blank means same date
1  cardnumber         ==> 5163 2256 7654 2235
  merchantid          ==>
  terminalid           ==>
  ..
|
.the ord field allows the user to pre-designate the sort-order of the
displayed transactions
```

The display title **Card Services Index** and the specific selection criteria input fields are determined by the administrator for this index database. (The *not-before* and *not-after* fields are always present).

Results are displayed as a scrollable list of selected transactions that satisfy the criteria:

The IQA selected transactions list display:

```
Card Services Index - selected transactions list                                1 of
7
command ==>

commands: pri(nt), sav(e), ta(g), un(tag) sort
line commands: s=select, p=print, t=tag, u=untag
lc cardnumber----- yymmdd-hhmm tranamount-- merchantid terminalid
s_ 5163225676542235      000822-1522          125.15 25962671   X49KN2
   5163225676542235      000822-1532          900.00 25962671   X49KN2
   5163225676542235      000822-1615         1000.00 25962671   X49KN2
  ..
```

A transaction can be selected by typing a *line command* in the input field at the left edge of the display and pressing the *ENTER* key.

S line command (view all data for this transaction):

The line commands defined for use on this display (**s (select)**, **p (print)**, **t (tag)** and **u (untag)**) are implemented via ECL scripts. This allows the installation to tailor the specific operation of a command and to define additional commands as required.

The primary commands defined for use on this display (**print**, **save**, **tag**, **untag**) are implemented via ECL scripts. This allows the installation to tailor the specific operation of a line command and to define additional line commands as required.

The transaction detail display looks like:

```

-----+-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7-----+-----8
Card Services Index - transaction detail                                     1 of      9
command ==>

commands: pri(nt), sav(e), ta(g), un(tag)

CARDNUMBER      : 5163225676542235
TRANDATE        : 000822
TRANTIME        : 1522
TRANAMOUNT       : 125.15
TERMINALID       : X49KN2
MERCHANTID       : 25962671
PROCCODE         : 003100
ACQUIRERID       : 0005602513
NAME_LOCN        : MICRO DIRECT SHEPPARTON 3630

```

The primary commands defined for use on this display (**print**, **save**, **tag**, **untag**) are implemented via ECL scripts. This allows the installation to tailor the specific operation of a command and to define additional commands as required.

This basic inquiry application can become the model for other specialised applications (the source code is provided as part of the software).

OMCS/RCX Utility Programs

OMCXEXTR

Sample JCL for OMCXEXTR

```
//HSCP      EXEC PGM=ECLX,
// PARM='OMCXEXTR script FILE(filename filetype) [options]'
//STEPLIB   DD DISP=SHR,DSN=your.ZSEN150.ASELOAD
//ECPLIB    DD DISP=SHR,DSN=your.ZSEN150.ASEECPS
//          DD DISP=SHR,DSN=your.script.library
//OMCXCTL   DD DISP=SHR,DSN=your.control.database
//SYSPRINT  DD SYSOUT=*
//
```

OMCXEXTR parameters

script the script library member name –or- * to indicate that the script is to be read from ddname SYSIN. The * option may be more convenient when developing a new script to avoid having to edit a separate file many times.

filename filetype identifies the database file containing the report or data file that the extract is to be performed on. Typically this will be an SDS file created by the OMCS Writer.

options processing options include:

Scheduling execution of OMCXEXTR by OMCXEXJS

(Still to come)

Example of OMCXEXTR Control Listing

(Still to come)

OMCXEXTR Messages

OMCX201e ddname *aa* not allocated

Application logic error or JCL error. The program attempted to open an index database and the supplied ddname *aa* was not allocated.

OMCX202e bad program parameters, rc=*nn*

An unsupported or erroneous parameter was specified. Correct the parameter error and restart the program.

OMCX203i processing file: *aa bb* with script *cc*

The program is processing SDS file *aa bb* which using script *cc*.

OMCX204i processing extract record set *aa*

The program is commencing the processing of extract record set # *aa*.

OMCX205i run parms: *aa*

This message displays the supplied program parameters.

OMCX206e Unrecognised term: "*aa*"

The program parameter term *aa* shown is unrecognised. Correct the parameter error and restart the program.

OMCX207e scriptname not specified, required

scriptname must be specified as the first parameter. Correct the parameter error and restart the program.

OMCX209i *yyyymmdd-hhmmss* START SCRIPT *aa* extract# *bb*

Script *aa* is being executed to create extract *bb*

OMCX210e load failed for module *aa*

The program cannot load the Run-Time Script Support module. Check the correct load module libraries are being used.

OMCX211I eof on i/p file after *aa* pages *bb* lines - rc=*cc*

The program has reached the end of the current input file after reading *aa* pages, *bb* lines. GETPAGE return code was *cc*.

OMCX212e file *aa bb* - no ASPLEVL and file not empty

Possible application logic error. The program is attempting to update the script execution register file. The file does not begin with a valid ASPLEVL= record. This situation requires diagnosis. Please contact ASE for support.

OMCX213e missing extract audit record ("X" rec)

Possible application logic error. The program is attempting to update a index file which has no valid Update History record. This situation requires further diagnosis. Please contact ASE for support.

OMCX214e file's update history shows more "E" records than the file contains, possible seq. error

Possible application logic error. The program is attempting to update a index file containing an invalid Update History record. This situation requires further diagnosis. Please contact ASE for support.

OMCX215I the program deleted %1 unaudited "E" records

The program has cleaned up what appears to be the residue of a previous index update that was cancelled or which failed to complete successfully for some other reason. The program continues processing.

OMCX216I skipping extract rec range *aa* already processed

The Update History record in the index file shows that the extract record range *aa* has already been processed. The program skips to the next record set.

OMCX217I file TRANDATE REGISTER last entry *aa* later than new entry *bb*, dump follows

Application logic error. The program is attempting to update the transaction date register (file TRANDATE REGISTER). There is no "D" record or the latest "D" record is for a later date *aa*. This situation requires further diagnosis. Please contact ASE for support.

OMCX218I file TRANDATE REGISTER structural error, dump follows

Application logic error. The program is attempting to update the transaction date register (file TRANDATE REGISTER). The file structure is not as the program expects to find it. This situation requires further diagnosis. Please contact ASE for support.

OMCX219I compile failed for script *aa*, rc=*bb*

User error.

OMCX223E There is no *aa* database defined in the settings register

The *aa* database must be defined in the settings register before you can proceed.

OMCX224E DBSTART failed for the *aa* database "*bb*"

The *aa* database (dataset *bb*) failed to start. Verify that the correct dataset name is specified in the settings register.

OMCX225w extract bypassed, no SDS file specified, rc=*aa*

The program is terminating because a report file was not specified, hence no extract can be performed. COND CODE 0004 is set to indicate this situation.

OMCX226w extract bypassed, script *aa* not compiled, rc=*bb*

The program is terminating because a compiled script was not available, hence no extract can be performed. COND CODE 0004 is set to indicate this situation.

OMCX227i Script execution took *aa* seconds CPU, rc=*bb*, *cc*

The script execution process has ended. This message documents the script return code *bb* and the number of CPU seconds *aa* taken by the script process. Any supplementary message text *bb* is shown on this message line.

OMCX228w extract# *aa* already present in extracts register

The program is attempting to update the script execution register. An entry already exists for this extract number *aa*. Another extract number will be assigned.

OMCX229e extract register record was missing, recreated

The program was unable to locate an existing extracts register record. This may indicate that a sequence error exists in the file EXTRACTS REGISTER. This situation must be investigated urgently. Contact ASE for support.

OMCX230e SIMULATE is in effect, wont create an EXTRACT# file

The program detected the SIMULATE keyword on the first OPTIONS statement in the script or in the PARM= string. An EXTRACT# nnnnnnnn file will not be created by this script execution.

OMCX231w extract ended rc=*aa*, *bb*

The script execution has terminated with a non-zero return code. Any supplementary message text *bb* is shown on this message line.

OMCX232i started RCX *aa* database *bb*

The RCX database *aa* is allocated to dataset *bb*. The program writes any extract file to the "work" database and looks for the source file to be indexed in the "reports" database.

OMCXHSP

Sample JCL for OMCXHSP

```
//HSP      EXEC PGM=ECLX,PARM='OMCXHSP'
//STEPLIB  DD DISP=SHR,DSN=your.ZSEN150.ASELOAD
//ECPLIB   DD DISP=SHR,DSN=your.ZSEN150.ASEECPS
//OMCXCTL  DD DISP=SHR,DSN=your.control.database
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
(control statements go here)
//
```

OMCXHSP parameters

None supported.

OMCXHSP control statement format and syntax rules

```
function operand1 [operand2 +      /* comment area
                   operandn]      /* comment area
```

- Function** begins in the first position of the first line of the statement;
- Operands** one or more spaces separate each operand from the preceding term;
- Comments** /* signals that the rest of this line is a comment area.
- Blank line** blank lines between statements are ignored.
- continuation** if the last non-blank character before the comment area is a plus (+) then a continuation of this statement is expected on the following line.

OMCXHSKP Control Listing

OMCXHSKP writes a Control Listing report via DDname SYSPRINT:

```
OMCXHSKP - Control Listing - os:MVS/XA sys:GSYS d:20070521 t:122531 l:050726
OMCX932I Started RCX control database your.omcsrxc.control.database
OMCX932I Started RCX work database your.omcsrxc.work.database
OMCX932I Started RCX index database your.omcsrxc.index.database
The current fields register entries are:
fld# field_name      type hlen slen properties...
  0                U         4    0
  1 DATE            D         4    0
  2 TIME            T         3    0
  3 CARDHOLDER     N        12    0 key field
  ..
////////////////////////////////////
first control statement image..
```

OMCXHSKP - Control Listing - .. heading line fields:

os:MVS/XA operating system upon which OMCXHSP was run (always MVS/XA for MVS/XA, OS/390 and z/OS)

sys:xxxx where xxxx is the SMFID of the system/LPAR

d:yyyymmdd where yyyymmdd is the date on which OMCXHSP was run

t:hmmss where hmmmss is the time at which OMCXHSP was run

l:yymmdd where yymmdd is the build date for OMCXHSP

OMCX932I Started RCX *type* database *dsname*

type indicates the OMCS/RCX database type; **control**, **work** or **index**.

dsname the corresponding dataset name.

the current fields register entries are:

```
fld# field_name      type hlen slen properties...
```

The content of the Fields Register is always shown at the start of the OMCXHSP Control Listing. This is a matter of forensic convenience, simply saving support staff from losing time while this information is obtained by other means.

fld# decimal field number, 1,2,.. 255. field# 0 is reserved for the date field, which is always present.

field_name the corresponding field name defined in the fields register.

type type code letter specifying how the data for this field is stored in the index. Values include: C, V, N, D, T, \$.

hlen the held-length in bytes.

slen the show-length in bytes.

properties shows special field attributes, eg "key field"

OMCXHSP functions: MAXCOND

MAXCOND nnnn

Sets a maximum allowable condition code that subsequent functions may return without the program terminating. MAXCOND statements may be used anywhere within the control statement file. If omitted, the maxcond value defaults to 0004.

OMCXHSP functions: EXTRACT

EXTRACT function extract#list

A group of functions are available for obtaining information about and manipulating files of extract data files.

Function a function to be performed upon extract register entries or upon extract data files. Supported functions are described below.

extract#list: **nnn [nnn nnn]**
 nnn:nnn
 ALL

Extracts are numbered from 1 onwards. The extract(s) to be processed by the function may be specified by number, by list of numbers, by range(s) of numbers, or by the keyword "ALL".

EXTRACT HOLD extract#list

Increments the HOLD count for the specified extract(s). An extract with a non-zero HOLD count will not be processed by the index update program (OMCXTDXU). The HOLD count will not increase above 9. See also EXTRACT RELEASE.

EXTRACT RELEASE extract#list

Decrements the HOLD count for the specified extract(s). An extract with a non-zero HOLD count will not be processed by the index update program (OMCXTDXU). The HOLD count will not decrease below 0. See also EXTRACT HOLD.

EXTRACT LIST extract#list

Lists the details from the EXTRACTS REGISTER for each extract file.

extract#	started----	ended--	#records	rc	1	updt	start-	ended--	rc	sdsfname
e00000154	040503:0103	+0:0106	01460242	00	0	040503:0112	+0:0137	00	FF100D9A	
e00000155	040504:0147	+0:0209	02048539	00	0	040504:0301	+0:0443	00	FF100D9A	

..and further to the right on each line is:

..sdsftype	script--(any parms)
..#0003453	FF100NEW()
..#0003454	FF100NEW()

This report shows essential details for each extract file:

- extract#** this number uniquely identifies the extract.
- started----** yymmdd:hhmm - shows date and time when extract started
- ended--** +d:hhmmsh – shows relative day (d) and time on that day (hhmm) when the extract ended. +0 means same day, +1 means next day and so forth.
- #records** the number of transactions extracted from the source file.
- rc** nn the extract return code
- h** the hold count. If this is > 0 then this extract will not be processed by the index update program, OMCXTDXU.
- updt start** yymmdd:hhmm - shows date and time when index update started
- ended--** +d:hhmmsh – shows relative day (d) and time on that day (hhmm) when the index update ended. +0 means same day.
- sdsfname** the filename of the extract source database file.
- sdsftype** the filetype of the extract source database file.
- script---** the name of the extract script member used by OMCXEXTR. Any parameters passed to the extract script at run time are shown within parentheses after the scriptname.

EXTRACT CLEAR_UPTS extract#list

Clears the *update timestamps* and *update return code* fields for specified extracts. The Index Update program (OMCXTDXU) sets started and ended timestamps and a return code field after it has completed processing each extract file.

```
updt start- ended-- rc
050603:0431 +0:0431 00
```

the update that processed this extract file started at 0431 on 3rd June 2005 and finished at 0431 on the same day. The update program returned cond code 0000 when processing this extract file.

OMCXTDXU will not reprocess an extract which already has these timestamps set. If it is determined that an extract must be reprocessed by OMCXTDXU then CLEAR_UPTS must be used first.

EXTRACT SETRC rc=nn extract#list

Sets the *extract return code* field in the extract register entry. An extract with a non-zero extract return code field will not be processed by the index update program (OMCXTDXU). See also EXTRACT HOLD and EXTRACT RELEASE.

EXTRACT COMPRESS extract#list [SIMULATE] OLDR=00 BIGR=05

Compresses those extract files which satisfy the selection criteria. Compression saves space in the work database and also saves space on archive datasets. Sets the EXTRACT return code field in the extract register entry. An extract with a non-zero extract return code field will not be processed by the index update program (OMCXTDXU). See also EXTRACT HOLD and EXTRACT RELEASE.

OLDR=nn specifies that only files older than **nn** days are eligible for selection.

BIGR=nn specifies that only files bigger than **nn** database blocks are eligible for selection.

OMCXHSP functions: DIAGNOSE**DIAGNOSE fn ft function options**

A group of functions are available for obtaining information from index files.

Fn ft The filename and filetype of the specific file, or generic group of files, to be processed. Supported functions are described below.

DIAGNOSE INDEXPRI yyyymmdd SHOWRANGES [DUMP] [FORMAT]

SHOWRANGES requests display of information about the source of each range of “E” (extract transaction) records contained in the specified index file. In the following example, two ranges are shown, along with the extract number from which each range came. The “Qrecstate” flag indicates that the OMCXTDXU program successfully completed the update of the corresponding index keys in this index file.

This file contains the following E rec ranges:

R	firsterec#	last_erec#	extract#	Qrecstate
R	2	2	00000154	Y
R	3	471	00000163	Y

Once a specific extract# file has been identified, the corresponding EXTRACTS REGISTER entry will identify the original source file from which the transactions were extracted.

“R” records contain formatting information for the fields contained within the corresponding range of “E” records. This formatting information is used by the Indexing API to ensure correct formatting of each field in the “E” records when the data is retrieved by a query.

DUMP specifies that the whole “R” record is to be shown in dump format.

FORMAT specifies that the Extract Field Descriptor Entries (EFDE) are to be formatted for each “R” record.

DIAGNOSE INDEXPRI yyyyymmdd SHOWREC [options] record#list

SHOWREC requests display of information about the specified record(s). Record types “E”, “Q”, “R” and “ASPLEVL” are recognized for index files. Record type “E” is supported further by recognition of the supported source of each range of “E” (extract transaction) records contained in the specified index file.

options may be any of:

DUMPE specifies that “E” records are to be shown in dump format, like this:

E rec # 49 dump:					
0000(+0000)	C5111700	00000001	03356235	10420015	E..... ..
0016(+0010)	61FFFFFF	FF064040	40400700	0000641F	/..... ..
0032(+0020)	00000064	08C9C9F0	F0F0F140	4009F1F1	II0001 .11
0048(+0030)	F1F2F2F2	F3F30A00	3100FF0C	56025102	122233..... ..
0064(+0040)	FFFF0B00	03500002	FF140210	FF150512	&.....

FORMATE specifies that “E” records are to be shown formatted, like this (only part shown):

```

E rec # 49 fields:
T=111700 L=1 3=N"3562351042001561FFFFFFFF" 6=C"      " 7=$1.00 31=$1.00
10=N"003100FF" 12=N"56025102FFFF" 11=N"0003500002FF" 20=N"0210FF" 21=
16=D20040512 17=T111701 18=V"0512      " 13=N"012F" 1
19=V"Melbourne      " 33=C"
32=C"

```

FNAMES specifies that “E” records are to be shown formatted by field.

```

E rec # 2269070 fields:
T=091218 L=5022968
CARDHOLDER      3=N"5602512364931969FFFFFFFF"
CURRENCY         6=C"USD "
TRANS_AMOUNT     7=$100.00
AUD_AMOUNT       31=$128.76
TERM_ID          8=C"TR0021 "
MERCH_ID         9=C"A-Frank'"
PROCCODE         10=N"102000FF"
ACQ_ID           12=N"430183FFFFFFFF"

```

record#list has the form:

```

nnn [nnn nnn]
nnn:nnn
ALL

```

The records to be processed by the function may be specified by number, by list of numbers, by range(s) of numbers, or by the keyword “ALL”.

DIAGNOSE INDEXPRI yyyymmdd SHOWKEY [options] nn:'keyvalue'

SHOWKEY requests display of information about record(s) associated with a specific key field value.

nn:'keyvalue'

nn specifies the decimal field number for the required key field. To determine which fields are key fields, and what the corresponding field number is, inspect the listing of the fields register produced automatically at the start of every OMCXHSKP control listing.

keyvalue specifies the key field value for the “Q” record(s) to be selected for processing according to the options specified. Enclose the value in apostrophes if the value includes spaces or non-alpha-numeric characters.

keyvalue may end in asterisk (*) to imply that generic or wildcard matching is required. The program locates the “Q” record(s) that best match the specified key. Operations performed are determined by the options specified.

options may be any of:

DUMP specifies that each “Q” record is to be shown in dump format, like this.

0000(+0000)	D8033562	35104200	1561FFFF	FFFF0000	Q../.....
0016(+0010)	00000031	00000345	00000659	0000093E	
0032(+0020)	00000C23	00000F08	000011ED	000014D2	K
0048(+0030)	000019BC				..

ENUM specifies that the numbers of the “E” records pointed to by the “Q” record are to be shown as decimal numbers.

The following example shows 16 “E” record numbers. Each of those “E” records contains the same value for field number 03 (x'5602512364931969ffffff'):

Q rec # 3251866 key section:					
0000(+0000)	D8035602	51236493	1969FFFF	FFFF0000	Q.™.©.šl.~.....
0016(+0010)	0022				..
...	...	...	...	...	...
2269070	2269071	2269072	2269073	2269074	2269075
2269076	2269077	2269078	2269079	2269080	2269081
2269082	2269083	2269084	2269085		

DUMPE specifies that “E” records pointed to by the “Q” record are to be shown in dump format, like this:

E rec # 49 dump:					
0000(+0000)	C5111700	00000001	03356235	10420015	E..... ..
0016(+0010)	61FFFFFF	FF064040	40400700	0000641F	/..... ..
0032(+0020)	00000064	08C9C9F0	F0F0F140	4009F1F1	II0001 .11
0048(+0030)	F1F2F2F2	F3F30A00	3100FF0C	56025102	122233..... ..
0064(+0040)	FFFF0B00	03500002	FF140210	FF150512	&.....

FORMATE specifies that “E” records pointed to by the “Q” record are to be shown formatted, like this (only part shown):

E rec # 49 fields:	
T=111700 L=1 3=N"3562351042001561FFFFFFFF" 6=C" " 7=\$1.00 31=\$1.00	
10=N"003100FF" 12=N"56025102FFFF" 11=N"0003500002FF" 20=N"0210FF" 21=	
16=D20040512 17=T111701 18=V"0512 " 13=N"012F" 1	
19=V"Melbourne " 33=C"	

When FNAMES is also specified, only the FNAMES formatting is shown.

FNAMES specifies that “E” records pointed to by the “Q” record are to be shown formatted by field, like this (not all fields shown):

E rec # 2269070 fields:	
T=091218 L=5022968	
CARDHOLDER	3=N"5602512364931969FFFFFFFF"
CURRENCY	6=C"USD "
TRANS_AMOUNT	7=\$100.00
AUD_AMOUNT	31=\$128.76
TERM_ID	8=C"TR0021 "
MERCH_ID	9=C"A-Frank'"
PROCCODE	10=N"102000FF"
ACQ_ID	12=N"430183FFFFFFFF"

OMCXHSPK functions: COMPRESS

COMPRESS INDEX [yyyymmdd[:yyyymmdd]] [SIMULATE] +
OLDRTHAN=nn BGRTHAN=nn

INDEX specifies that index files are to be selected for compression. Only primary index files will be considered. Secondary index files are never compressed.

Yyyyymmdd[;yyyymmdd] an optional term that specifies either a particular file - or- a range of files, to be selected for compression.

OLDRTHAN=nn specifies that only files older than **nn** days are eligible for selection. **OLDR=** is a supported abbreviation.

BGRTHAN=nn specifies that only files bigger than **nn** database blocks are eligible for selection. **BGR=** is a supported abbreviation.

SIMULATE specifies that the program is to operate normally except that it will not actually compress selected files.

COMPRESS EXTRACTS [SIMULATE] [OLDRTHAN=nn] [BGRTHAN=nn]

EXTRACTS specifies that EXTRACT# files are to be selected for compression.

OLDRTHAN=nn specifies that only files older than **nn** days are eligible for selection. **OLDR=** is a supported abbreviation.

BGRTHAN=nn specifies that only files bigger than **nn** database blocks are eligible for selection. **BGR=** is a supported abbreviation.

SIMULATE specifies that the program is to operate normally except that it will not actually compress selected files.

OMCXHSPK control statement example

```
//SYSIN DD *
diagnose INDEXPRI 20040512 + /* specify the file
        ENUM      + /* show E rec numbers in Q recs
        dump      + /* dump Q recs
        dumpe     + /* dump E recs
        FORMATE   + /* format E rec fields
        FNames    + /* show E rec field names
        03.3562351042001560* + /* a key field
        03.3562351042001562* /* another ..
//
```

In this example the program is asked to select two key records (Q records). The key field used to locate the each Q record is field number 03. The key value is separated from the field number by a period (.). In this case each key value is terminated by asterisk (*) to indicate that if an exact match is not found, the closest preceding Q record is to be selected.

After showing the Q record in dump format (DUMP) as well as a list of associated E record numbers (ENUM), the program is to format each of the E records (FORMATE) showing each data field by name (FNAMES).

OMCXTDXU

Sample JCL for OMCXTDXU

```
// HSKP      EXEC PGM=ECLX,
// PARM='OMCXTDXU EXT#=extract# [options]'
// STEPLIB DD DISP=SHR,DSN=your.ZSEN150.ASELOAD
// ECPLIB DD DISP=SHR,DSN=your.ZSEN150.ASEECPS
// OMCXCTL DD DISP=SHR,DSN=your.control.database
// SYSPRINT DD SYSOUT=*
//
```

OMCXTDXU parameters

EXT#=extract# this parameter tells OMCXTDXU which extract files it may consider for processing. It is either a decimal number or “*”.

If extract# is “*” (asterisk) then the program will process all extracts that have not yet been processed by OMCSTDUX.

OMCXTDXU will ignore extract files where the extract register entry shows a non-zero extract return code (set by OMCXEXTR) or a non-zero hold count.

Refer to the description of the OMCXHSKP utility for functions that list and manipulate extract register entries.

options The following optional parameters are supported:

MAXTIME=nnnn When EXT#=* has been specified the program will continue to process all unprocessed EXTRACT# files and if there are a lot of unprocessed extracts this can result in a job that runs for an elapsed time that is long enough to cause operational difficulties. The MAXTIME= parameter specifies the length of time after which the program must stop selecting and processing EXTRACT# nnnnnnnn files. The number nnnn must be in the range 1 to 1439 (there are 1440 minutes in one day).

Note that the program only makes this check after it completes the processing of each “EXTRACT# nnnnnnnn” file.

DIAG The program will display some additional diagnostic information. This option should only be used when requested by ASE support staff.

TRACE Parts of the program's execution will be traced in detail. This option should only be used when requested by ASE support staff.

Example of OMCXTDXU Control Listing

(Still to come).

OMCSTDUXU Messages

OMCX301e ddname *aa* not allocated

Application logic error or JCL error. The program attempted to open an index database and the supplied ddname *aa* was not allocated.

OMCX302e bad program parameters, rc=*aa*

An unsupported or erroneous parameter was specified. Correct the parameter error and restart the program.

OMCX303i processing file: *aa bb*, ASPLEVL rec follows:

The program is processing extracts file *aa bb* which was created by script *cc*.

OMCX304e omcxiusr.init failed, rc= *aa*

The program encountered an unexpected return code *aa* from the service routines module init routine.

OMCX305i run parms: *aa*

This message displays the supplied program parameters *aa*.

OMCX306e Unrecognised term: "*aa*"

The program parameter term *aa* shown is unrecognised. Correct the parameter error and restart the program.

OMCX307e extract# not specified, or invalid

Either **EXTRACT#(nnnnnnnn)** or **EXTRACT#(*)** must be specified in the program parameters to identify which extract file is to be processed. **EXTRACT#=** may be used in place of **EXTRACT#(..)**. **EXT#** is a valid abbreviation of **EXTRACT#**. When **"*"** is specified instead of the extract number, the program will search the **EXTRACTS REGISTER** file to find the next unprocessed extract file (**EXTRACT# nnnnnnnn**) if any. Correct the parameter error and restart the program.

OMCX308i *aa* ended, rc=*bb cc*

Information message. The process *aa* has ended returning return code *bb*. If SHOWT was in effect then CPU and EXCPs counts are shown.

OMCX309e missing D record, no extracted fields attributes

Data specification error or file format error. The program did not find a "D" record as the second record of the extracts file. This may be caused by the user specifying the name of an invalid extracts file.

OMCX310e sort failed, terminating.

Possible operator or user error. The program was unable to sort data in memory. Review the REGION= value specified, increase it and rerun. If the problem persists, please contact support.

OMCX311e reached eof on input extract file

The program has reached the end of the current extracts file.

OMCX312e error, cannot update compressed file *aa bb*

Probable operations or user error. The program is attempting to open file *aa bb* to update it. The file has been compressed and cannot be updated. Uncompress the file and rerun the job.

OMCX313E build resident EXTS structures failed, rc=*aa*

Increase the region size and rerun the job. If the problem persists, please contact ASE for support.

OMCX314I there are no more unprocessed extract files

The program was invoked with EXTRACT#=* and there are no more unprocessed extract data files as represented by records in the EXTRACTS REGISTER file with an update start date:time field of ____:____.

OMCX315I file *aa bb* will be updated

The program has selected file *aa bb* to be updated.

OMCX316e locate error, date:*aa* time:*bb* E rec#:*cc*

Probable logic error. While processing data for date *aa* the program was unable to position to record number *cc*. Please contact ASE for support.

OMCX317e insert error, E rec date:*aa* time:*bb* rec#:*cc*

Probable database space shortage. While processing data for date *aa* the program was unable to insert E record number *cc*. Please contact ASE for support.

OMCX318i inserting R rec for unaudited E recs *aa*

The program has detected that the last E rec in the index file does not correspond with the last R record. The program will insert an R rec for the unaudited range of E records.

OMCX319e update-record failed, rec#=*aa* nderec#=*bb*

Program logic error. Please contact ASE for support.

OMCX320e OPEN failed for index file during Q rec update

The program could not open the index file to be updated. Please refer to prior message(s) for more detail

OMCX321e LOCATE failed during Q rec update

The program encountered a return code > 8 from a \$DBAM LOCA macro call. Please contact ASE for support.

OMCX322e UPDR failed during Q rec update

The program encountered a return code > 0 from a \$DBAM UPDR macro call. Please contact ASE for support.

OMCX323e IRAC failed during Q rec update

The program encountered a return code > 0 from a \$DBAM IRAC macro call. Please check that the index data database is not full. If there is ample free space, please contact ASE for support.

OMCX324e CLOSE failed during Q rec update

The program encountered a return code > 0 from a \$DBAM CLOS macro call. Please contact ASE for support.

OMCX325e unknown rc *nn* from Q rec update

The program encountered a return code > 0 from the UPDQ service routine. Please contact ASE for support.

OMCX326e unspecified error sorting keys for Q rec update

The program encountered a return code > 0 from the \$SORT service routine. Please contact ASE for support.

OMCX327i memory area... size...

The program shows the sizes of several memory areas whose sizes depend upon how many Extract records are to be processed and the total size of the key fields they contain.

OMCX328I extract# *aa* not found in extracts register file

The program was told to process file EXTRACT# *aa* but there is no matching entry for this extract# in the EXTRACTS REGISTER file. Correct the extract# and rerun the job, -or- use EXTRACT#=* to process all unprocessed extract data files.

OMCX329E extract process rc not 00, 04, update bypassed for *aa*

The program was told to process file EXTRACT# *aa* but the EXTRACTS REGISTER file entry shows that the extract program ended with an unacceptable return code. The program terminates. Review the JCL listing and report from the extract job to see what the problem was.

OMCX330E extract *aa* is held, cannot process further

The program selected file EXTRACT# *aa* but the EXTRACTS REGISTER file entry shows that the extract file is HELD. The program will not process this file. Contact the application administrator for further detail about the reason for the HELD state.

OMCX331i error diagnostics, save for support

The program has encountered an unexpected error during database file updating. The storage dumps that follow may be required for problem diagnosis.

OMCX332E extract *aa* duplicate for file *bb cc*

The program found a duplicate extract register entry *aa* for a SDS file *bb cc*. The program displays both extract register entries and then terminates.

OMCX333i there are no eligible extracts to process

The program found no unprocessed extract register entries with extract rc=00 and held=0.

OMCX334E MAXTIME *aa* not decimal 0:1439

The program detected an invalid MAXTIME value.

OMCX335E MAXTIME limit prevents further processing

The program found that the calculated time to cease processing extract files had been reached.

OMCX336E EXTRACTS REGISTER error, cannot find *aa* entry

The program cannot locate an entry for extract *aa* in the EXTRACTS REGISTER file.

OMCX398i program %1 terminating with COND CODE %2 %3

Program *aa* is terminating with condition code *bb*.

OMCX399e open failed for file *aa bb rc cc*

Possible application logic error. The program could not open the file *aa bb*. Refer to a preceding message that should give more specific information about the cause.

The Index Database API

API Introduction

The OMCS/RCX software includes an API to allow querying of the index database.

Each index “database” actually consists of a set of several databases: **control**, **index**, **work** and **report**. Before the process of extraction and index updating can begin the index database must be configured using the RCX administration application. After that the query application API calls can be used to extract information about the index and index data itself.

The basic API functions are:

index_session_start	(starts a session with an index database)
index_fields_query	(returns information about index data fields)
index_data-select	(retrieves index items that satisfy criteria)
index_data_release	(releases index_data_select resources)
index_session_stop	(ends a session with an index database)

Outline of a typical API application:

The application must be written in ECL to make use of the API.

A typical API application will use API calls in the order shown below:

```
index_session_start  /* start a session with an index database
index_fields_query   /* obtain information about fields in the data
index_data_select    /* specify criteria and retrieve matching items
(process selected data)
index_data_release   /* release resources and data that matched criteria
index_session_stop   /* close the session with the index database
```

Specific invocation and response details of each API call are provided in the reference section following.

API calls reference

Starting a session with an index database

The index database session is started using the `index_session_start` function. `index_session_start` returns a session number, a positive number which is used to identify a specific index database session and which will be used in other API calls.

```
index_session_start +
    filename +      /* filename part of database file-id
    filetype +      /* filetype part of database file-id
    ddname=ddname + /* ddname alloc. to the index database
    returns(rc +    /* return code variable
                sess# + /* index session number
                lmsg)  /* error message variable
```

filetype and **filename** together these two form the file-id of the specific database file containing the index to be processed.

ddname=ddname this defines the ddname which is allocated to the database dataset.

returns(..) variables:

`index_session_start` sets three response variables in the invokers variable pool:

rc The first **returns(..)** variable is a return code indicating success or failure of the function. This variable will be **0** if the function worked, **>0** if errors were encountered. If **>0** then the third **returns(..)** variable will contain explanatory text. See **lmsg** below.

Sess# If `index_session_start` is successful then the second **returns(..)** variable contains the session number. The session number is required by other API calls. Multiple index databases can be open at the same time and a different session number will be used for each session started. Session number will be zero if `index_session_start` is not successful.

lmsg if the function failed then the third **returns(..)** variable will contain explanatory message text.

Selecting items that match criteria

The index database may be queried using the `index_data_select` function:

```
index_data_select +
    &sess# +          /* session number
    &handle +          /* selection handle -or- ? if new
    cardnumber=(5163....,5163....) + /* values for a key field
    merchantid=(....) + /* multiple key fields are allowed
    terminalid=(....) + /* ..
    datetime=(from:to) + /* yyyyymmdd[-hhmmss]:yyyyymmdd[-hhmmss]
    .. +
    items=(from,for) + /* which items to retrieve this call
    show=(#/*/name1,name2,...) + /* elements to show for each item
    format=(vs/cs/bn) + /* presentation format for elements
    todd=ddname + /* write data to flat file (optional)
    todb=fileletter + /* write data to d/b file (optional)
    returns(rc + /* rc>0 if errors, see lmsg
        handle + /* selection handle variable
        #items + /* number of unretrieved items
        lmsg) + /* error message if rc>0
        itemlist) /* item data return variable (optional)
```

sess#

The *sess#* variable specifies the value returned by a previous **index_session_start** function.

handle

If handle is “?” then this is regarded as an initial selection call. **Index_data_select** chooses a number to return as a handle. This chosen handle value is returned as the second **returns(..)** variable.

If the handle is not “?” then this is regarded as a subsequent selection call. The value must be a currently valid handle number returned by an earlier initial selection which has not yet been discarded using the **index_data_release** function.

fieldname=valuelist

The *index_data_select* call requires a search argument to be provided for at least one fieldname. The syntax supported is to use the fieldname as a keyword and provide the search argument as the value term. If there are multiple arguments then they must be comma-separated and enclosed in parentheses.

At least one of the specified fieldnames must have the KEY attribute in its definition.

Example:

cardnumber=(5163222500253072,51636205210022065)

datetime=(from:to)

The datetime= operand may be used to specify a date and optional time range for the selection. The from and to values must be specified in the format yyyyymmdd-hhmmss.

Example:

datetime=(20020501:20020531)

items=(from[,for])

index_data_select identifies a set of data items that meet the specified criteria. The **items=** operand identifies which items(s) are to be returned by this *index_data_select* call. Subsequent *index_data_select* calls may be made with **items=** specified to retrieve other selected items. Note that the **#items** variable is always set to the difference between total number of items selected and the highest item number retrieved by this call. This can be used by a program to control an iterative process to retrieve selected items one at a time until all have been retrieved, at which time the **#items** variable will have a value of zero.

If omitted, **items=** defaults to "1,1"

If **items=0**, or **items=(n,0)** was specified then the *index_data_select* function returns no items. In this case the **show=** and **format=** operands may be omitted. The programmer may simply want to know how many items met the specified criteria, without wishing to retrieve any. In any case, the number of items selected is always returned in the third **returns(..)** variable.

If the **for** value is specified as "*" then the rest of the selected items are retrieved.

show=elementnamelist

Each selected item consists of a number of data elements. The **show=** operand may be used to specify which data elements are to be returned ("shown") for each item retrieved, and optionally, the formatting for the returned data itself. Specify the corresponding fieldnames, separated by commas and enclosed in parentheses.

show=* specifies all elements are to be returned for each item retrieved. The element data values are ordered according to the element name list returned by the **index_field_query** call.

show=# specifies that the *itemid* element is to be returned for each item retrieved.

Examples:

show=(cardnumber,amount,time,terminalid)

The values of the four elements shown are to be returned for each item.

show=#

Itemid, a unique internally maintained number assigned to each item in the index database, is the field to be returned. The default presentation for *itemid* is a character string of decimal digits, like 123456789.

When *itemid* is the only element to be shown, ie. **show=#** is coded, then the format of the returned number may be specified via the **format=** operand. **Format=An** (where n=4/5/6/8) for example, may be more convenient for subsequent processing by some applications.

Using the optional data element formatting controls:

The programmer may specify format controls for individual data elements by appending a parenthesised list to the end of the corresponding field name in the **show=** list.

Each format item consists of a **type code letter** followed by an optional length:

C8	left-justified character string, 8 bytes, left-filled with blanks
R8	right-justified character string, 8 bytes, right-filled with blanks
Z9	right-just. character string, 9 bytes, left-filled with zeros
V	vstring format &vs(element) ie. prefixed with a 2-byte binary length
Vn	vstring format &vs(element) - data truncated to n bytes
Bn	binary data format where n , the length in bytes, may be 1,2,4,6 or 8.

Example:

show=(cardnumber(c20),amount(r9),trantime(c4),terminalid(c7),#

list=(cs/vs)

The format of the retrieved data elements for an item can be specified as a comma-separated list or a variable-string list.

list=cs specifies comma-seperated list format, eg:

"5163.....", "23.50", "15:16:12", "ANZ1234"

When data values contain embedded blanks or other characters likely to confuse subsequent parsing of the comma-sparated list, such as comma itself, then the data value will be enclosed in single apostrophes. If the data value contains single apostrophes, but not double apsotrophes, then it will be enclosed in double apostrophes. If the data value contains double apostrophes, but not single apsotrophes, then it will be enclosed in single apostrophes. If the data value contains both single and double apostrophes then it will be presented as a string of hexadecimal digits, enclosed in single apostrophes and prefixed with the letter "x".

list=vs specifies that data elements are returned as a list consisting of a two-byte binary length followed by the data element value, followed by the next two-byte binary length and so forth. Application programs will usually find this format faster to process than a comma-separated list and parsing the list is not sensitive to the data content.

list= will be ignored if coded when **show=#** is specified.

todd=ddname

specifies that the retrieved items are to be written as seperate records to the sequential file designated by the ddname. This item is mutually exclusive with the **todb=** keyword option. See also the description of the *itemlist* returns variable.

todb=fileletter

specifies that the retrieved items are to be written as seperate records to the database file designated by the specified fileletter. This item is mutually exclusive with the **todd=** keyword option. See also the description of the *itemlist* returns variable.

returns(..) variables:

index_data_select returns several variables to the invoker. Note that the names shown are arbitrary choices, the values assigned are positionally dependent.

rc **0** if no errors encountered, **>0** if errors encountered. If the return code is **>0** then the fourth **returns(..)** variable (shown as **lmsg** below) will contain explanatory text.

handle a number which identifies the results of this selection. Selection results are kept until the programmer uses **index_data_release** to release them, or until **index_stop** is executed. The handle number may be passed into subsequent executions of **index_data_select** to prevent the re-evaluation of selection criteria. The programmer may choose to, or need to, execute **index_data_select** multiple times to retrieve all items identified by the initial **index_data_select**. See the description of the **items=** operand above.

#items the total number of items satisfying the selection. A single **index_data_select** may select many items. The programmer may choose to execute **index_data_select** again, specifying the handle number returned by the initial **index_data_select**, and an item number which is incremented each time to cause retrieval of the next item, until the item number reaches the value of the **#items** variable.

lmsg when the return code is **> 0** then the fourth **returns(..)** variable contains explanatory message text

itemlist When **todd=** and **todb=** are not specified, **index_data_select** sets the fifth **returns(..)** variable to the value of the the selected item(s). The string of data for each item is prefixed with a two-byte binary length field. The format of the data within that string is determined by the **show=** and **list=** keyword operands or their defaults if omitted.

Discarding a set of selected items

The set of items selected by an initial **index_data_select** will be retained until the programmer discards them, releasing either **index_data_release** or **index_stop** is executed. All resources utilised to maintain the selection details are released, including the handle number.

```
index_data_release +
    sess#    +           /* session number (see "index_start")
    handle   +           /* handle number for a current selection
    returns(rc lmsg) /* variables returned
```

sess#

The **sess#** variable specifies the value returned by a previous **index_session_start** function.

handle

A currently valid handle returned by a previous **index_data_select** function for this index database.

returns(..) variables:

index_data_release returns variables to the invoker.

rc The first **returns(..)** variable is a return code indicating success or failure of the function. This variable will be 0 if the function worked, >0 if errors were encountered.

lmsg if the function failed then the second **returns(..)** variable will contain explanatory message text.

Ending a session with an index database

An index database session is stopped using the **index_stop** function. Any current selections not yet discarded by **index_data_release** will be discarded by **index_stop**.

```
index_stop +
      ddname=ddname + /* ddname alloc. to the index database
      returns(rc lmsg) /* variables returned
```

ddname=ddname this defines the ddname which is allocated to the database dataset.

returns(..) variables:

index_stop returns variables to the invoker.

rc The first **returns(..)** variable is a return code indicating success or failure of the function. This variable will be 0 if the function worked, >0 if errors were encountered.

lmsg if the function failed then the second **returns(..)** variable will contain explanatory message text.

Querying names & attributes of fields defined in an index database

```
index_fields_attributes_query +
      &sess# + /* session number
      elementname/* + /* name of element/*=>all
      show=(names/attributes) +/* what is to be queried
      [todd=ddname +] /* write data to flat file
      [todb=fileletter +] /* write data to d/b file
      returns(rc + /* rc>0 if errors, see lmsg
      lmsg + /* error message if rc>0
      namelist) /* used if todd/todb not specifed
```

Publication number: OM03-0226-1

This manual serves as a reference source for users of the OMCS/RCX product. Please use this form to comment on the publication in terms of its organization or content with the understanding that ASE may use or distribute the information that you supply in any way that it believes appropriate without incurring an obligation to you.

If you require further copies of this publication please order them through your ASE representative. If you require assistance in the installation, configuration or use of the OMCS/RCX product please contact ASE for support.

Suggested aspects that might be addressed by your comments include:

Clarity Accuracy Completeness Organization Examples Legibility

Please give your name and address if you require a reply:

name:

email:

Address:

Occupation:

Comments:

